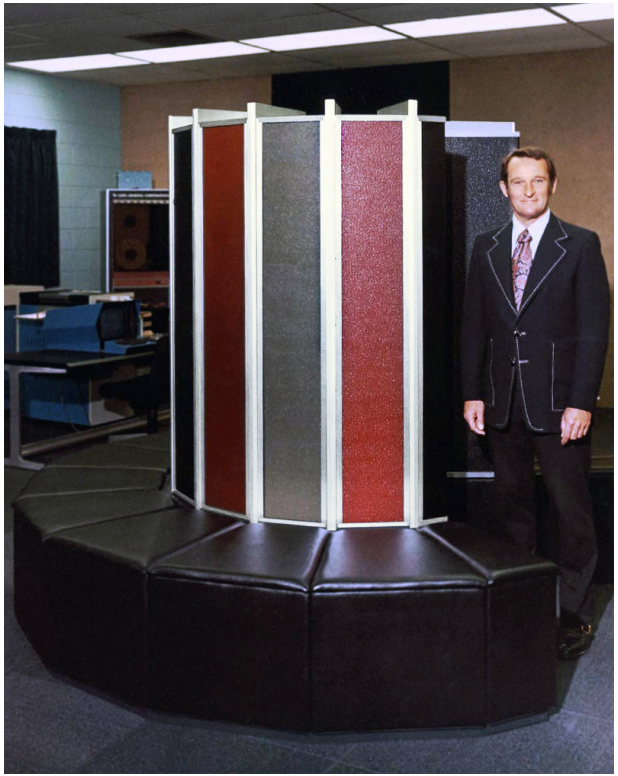
Data Science at Supercomputer Scale

Mike Ringenburg, Cray

NERSC Big Data Summit 2018

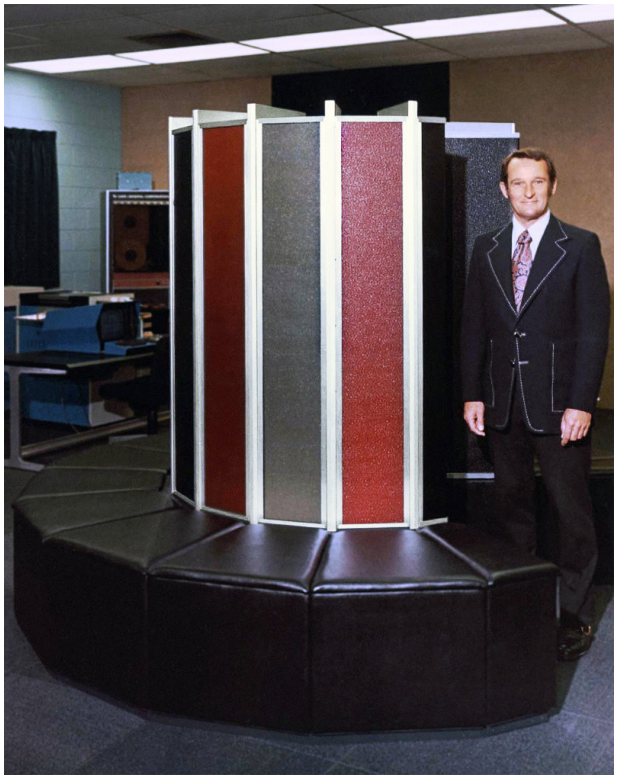
Cray, Supercomputing and Data Science

- Cray: a supercomputing pioneer since 1976



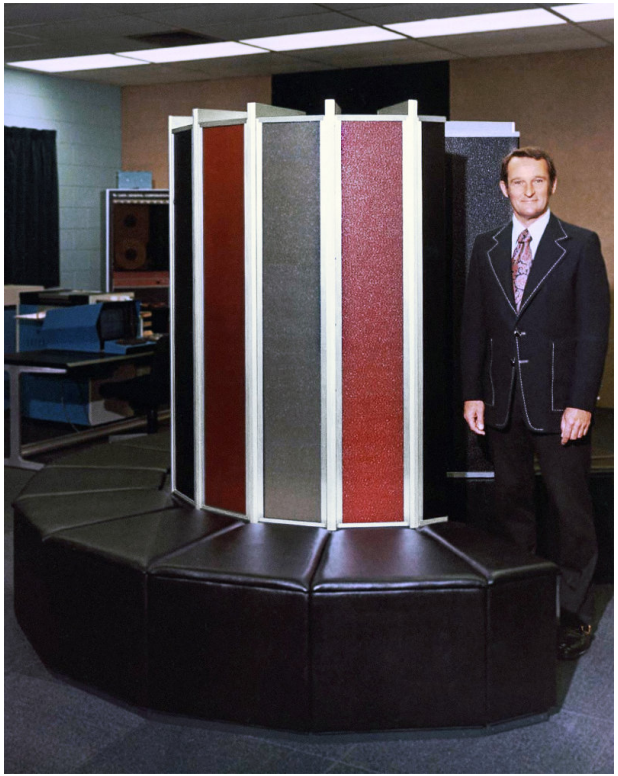
Cray, Supercomputing and Data Science

- **Cray: a supercomputing pioneer since 1976**
- **Supercomputing is changing**
  - Increasingly seeing need for data science in scientific computing and other traditional HPC environments
    - Scientific use of AI
    - Training at large scale
  - Collaborating with NERSC BDC, Intel, and other partners to understand landscape and develop solutions



Cray, Supercomputing and Data Science

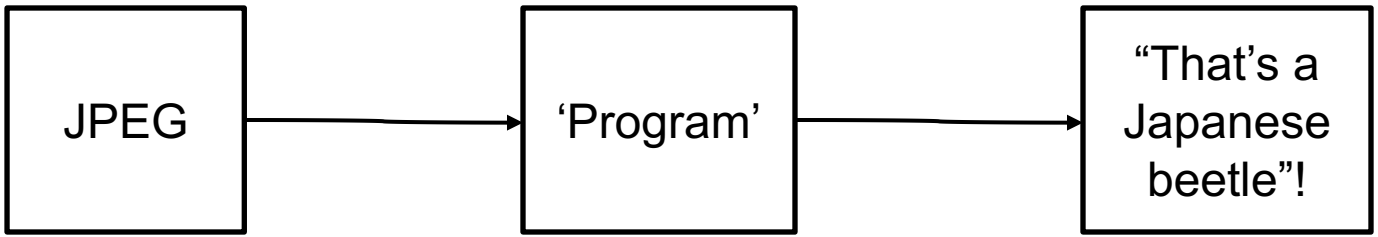
- **Cray: a supercomputing pioneer since 1976**
- **Supercomputing is changing**
  - Increasingly seeing need for data science in scientific computing and other traditional HPC environments
    - Scientific use of AI
    - Training at large scale
  - Collaborating with NERSC BDC, Intel, and other partners to understand landscape and develop solutions
- **Current efforts**
  - Urika-GX analytics platform
  - Urika-XC and Urika-CS software stack
  - **The Cray PE ML Plugin**
  - **Alchemist**
- **Where we are headed**
  - **End-to-end workflows**
  - **The convergence of AI and Simulation**



Scaling Deep Learning

What is Deep Learning: A Specific Example

- An organic gardener is building a robot in his garage to recognize the 10 insects found in his garden, and decide which ones to kill with a laser
- The robot will have a camera, and will capture JPEG files of the insects
- The robot needs a 'program' to classify each JPEG according to which of the 10 kinds of insect was photographed



Inputs & Outputs

- **Our input is a JPEG**
  - 224x224 pixels, 3 colors → a 224x224x3 element vector of the pixel values
- **Our output is a classification**
  - One of 10 categories → a 10 element vector with a "1" in the position representing the category to which the image belongs

How many "IF" statements will we need to figure out that a bunch of pixel values is a Japanese beetle?

Bruce Marlin
(CC BY 3.0: Attribution 3.0 Unported)

This is an Artificial Intelligence Problem

- If you can't get the output from the input with a bunch of loops and conditionals, it's AI
- But, if that won't work, how can we do it?

- Hint #1: Any mapping of inputs to outputs is a function
- Hint #2: A function can be approximated using a (good) approximating function

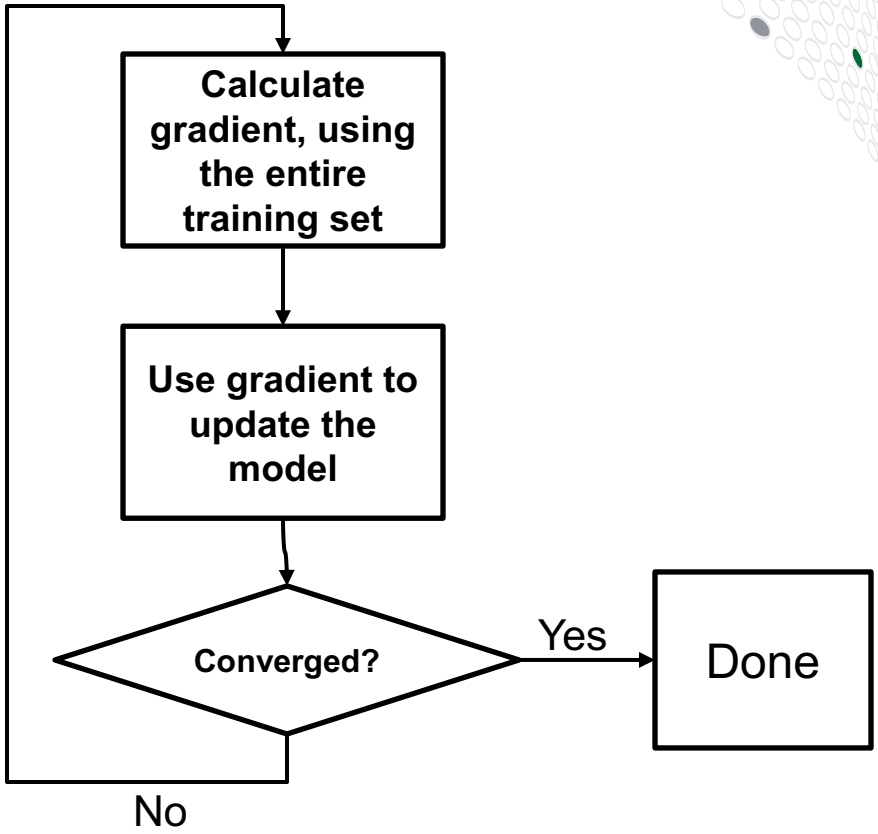
An Approximating Function

- **How can we determine a good approximating function?**
  - Choose its form (linear, polynomial, …)
  - Minimize the overall error at a finite number of inputs with known outputs - - **fit the curve**
    - We have to find the values of the free parameters of the function that minimize the error – it doesn't matter how we do it

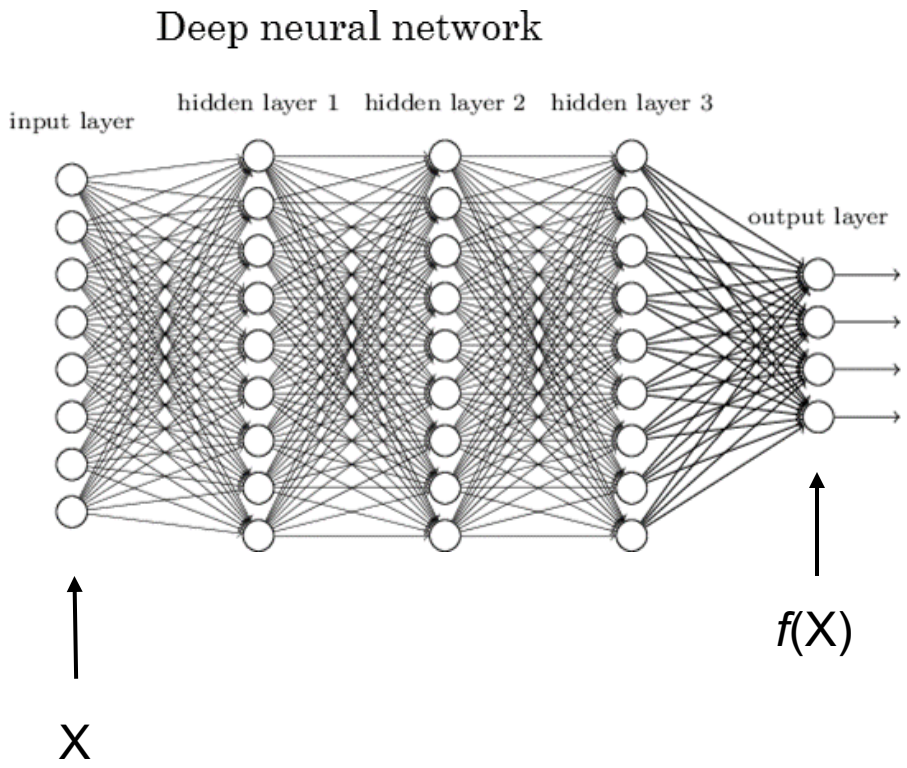
Fitting the curve is a lot like *training* the function to know the answer for arbitrary inputs

Training via Gradient Descent

- **We want to approximate y=*f*(x)**
  - Find a function that maps a set of inputs to a set of outputs, to some level of accuracy
- **We know $y_i=f(x_i)$, for i=1,N**
- **Iterate:**
  - First iteration only: initialize the free parameters of *f*
  - Calculate error (over N known points)
  - Calculate gradient of error, as a function of the free parameters of *f*
  - Adjust the free parameters of *f* a 'small' distance in the direction of negative of error gradient
  - Assess convergence & stop when 'good enough'



A Really Useful Kind of Function



- **This image shows a *deep neural network***
  - An approximating function, with free parameters called *weights* and *biases*
  - Deep networks have been found to be especially powerful
  - Neural networks can approximate any continuous function arbitrarily well

HPC Attributes of Deep Learning

- **Today we'll hear examples of a number of deep learning applications that require HPC-scale resources to train**
  - Cosmology
  - Climate
  - Life Sciences

- **DL training is a classic high-performance computing problem which demands:**
  - Large compute capacity in terms of FLOPs, memory capacity and bandwidth
  - A performant interconnect for fast communication of gradients and model parameters in order to scale up
  - Parallel I/O and storage with sufficient bandwidth to keep the compute fed at scale

Parallelization Techniques

- **Data Parallelism**
  - Divides global mini-batch among processes
  - Two methods for this:
    - Synchronous: single model (possibly replicated across all processes) updated with globally averaged gradients every iteration
    - Asynchronous: processes provide gradients every iteration but are allowed to fall out of sync from one another. Processes each have their own model that may or may not be the same as any other process
- **Model Parallelism**
  - Single model with layers decomposed across processes
  - Activations communicated between processes
- **Synchronous data parallel approach is most common, today…**

Data Parallelism - Collective-based Synchronous SGD

- Data parallel training divides a global mini-batch of examples across processes
- Each process computes gradients from their local mini-batch
- Average gradients across processes
- All processes update their local model with averaged gradients (all processes have the same model)

Algorithm 1 Sync-SGD algorithm

for $0 \leq step < max\_steps$ **do**

$G_{local} \leftarrow$ COMPUTE_GRADIENTS(mini batch)

$G_{global} \leftarrow 1/N_{ranks} \times$ ALLREDUCE(G_{local})

APPLY_GRADIENTS(G_{global})

end for

Compute intensive

Communication intensive

Typically not much compute

- Not shown is the I/O activity of reading training samples (and possible augmentation)

Distributed TensorFlow

- **TensorFlow's native method for parallelism uses ClusterSpec API and gRPC layer**
- **Can be difficult to use and optimize. User must specify:**
  - hostnames and ports for all worker and parameter server processes
  - # of workers
  - # of parameter server processes
  - Chief process of workers
- **Number of parameter servers (PS) processes to use is not clear**
  - Too few PS results in many-to-few communication pattern (very bad) and stalls delivering updated parameters
  - Too many PS results in many-to-many communication pattern (also bad)
- **Users typically have to pick a scale and experiment for best performance**

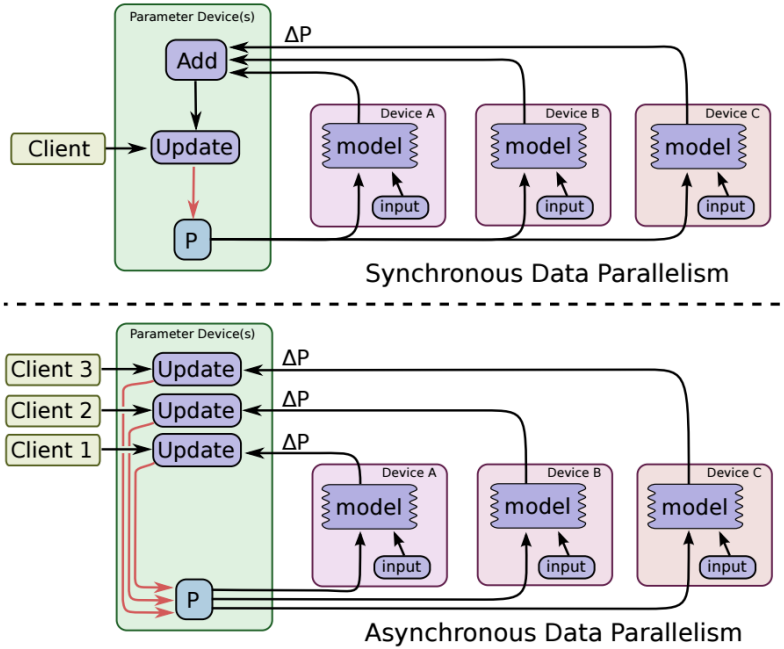
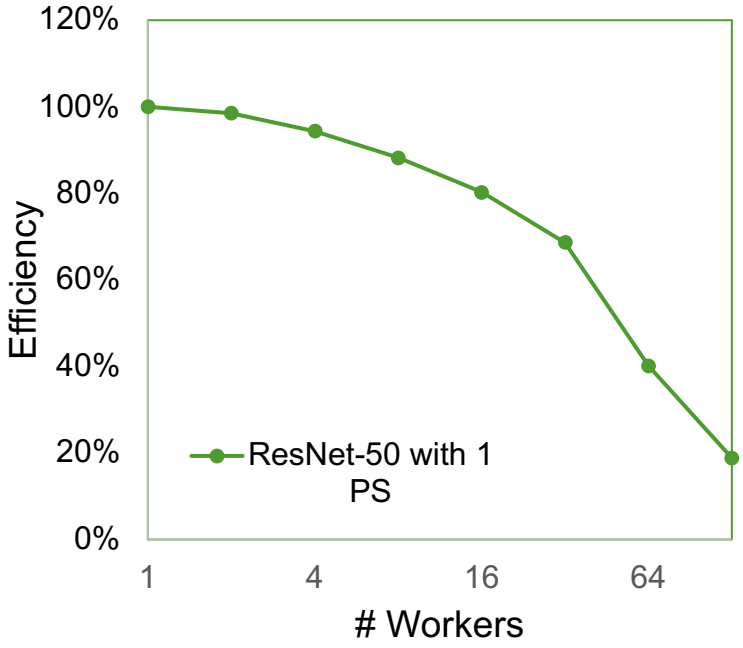
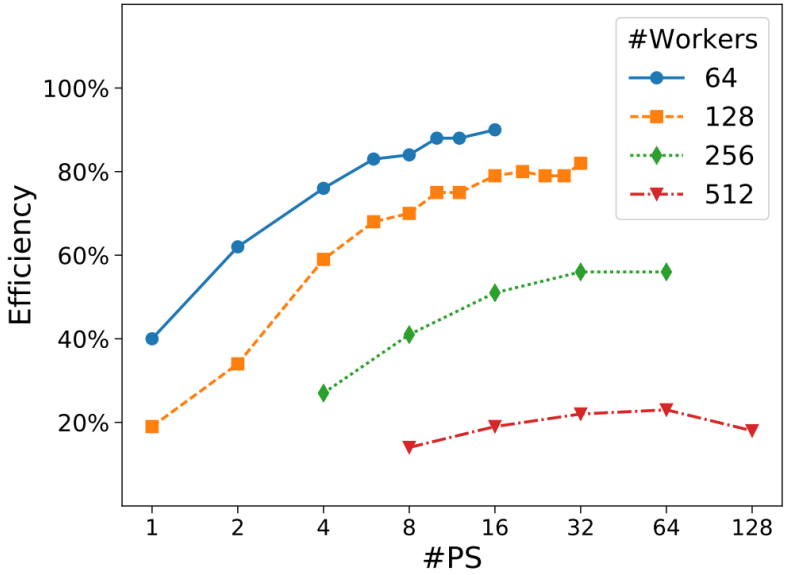


Figure 7: Synchronous and asynchronous data parallel training

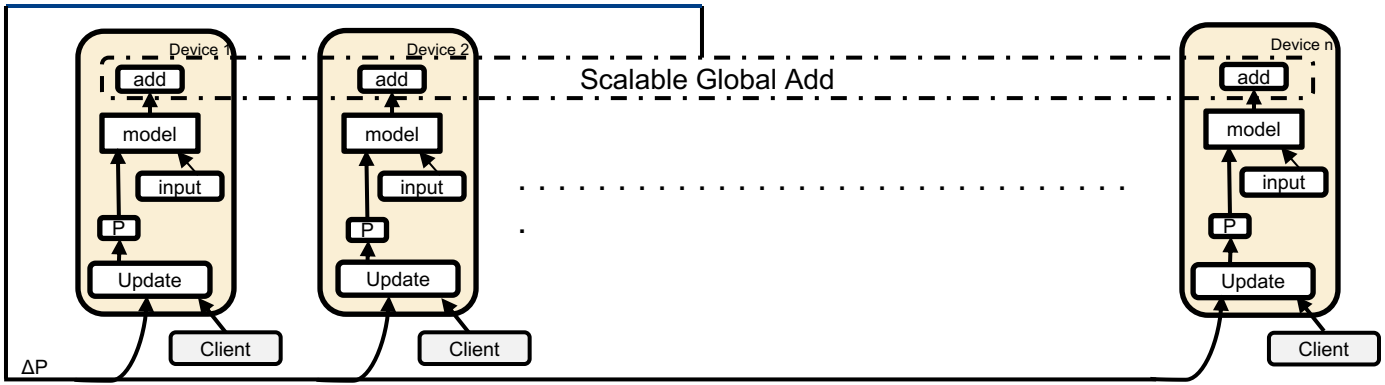
Distributed TensorFlow Scaling on Cray XC40 - KNL





From Mathuriya et al. @ NIPS 2017

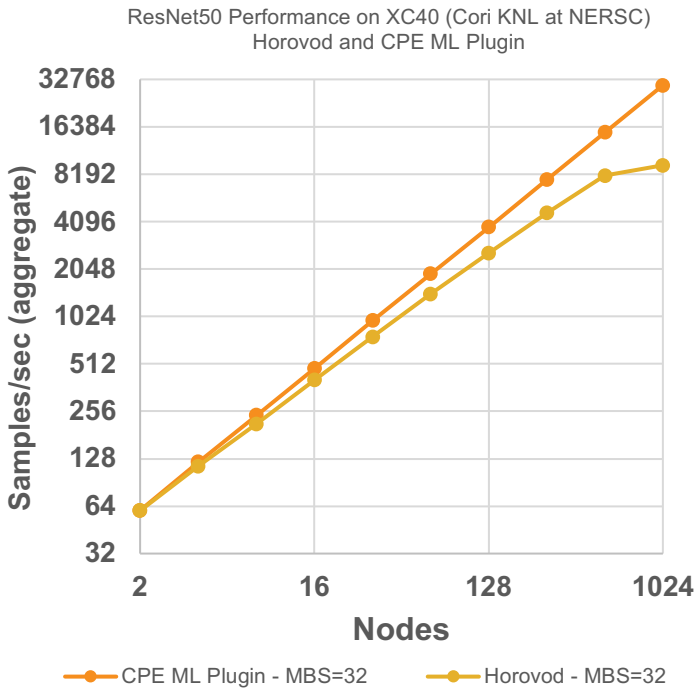
Scalable Synchronous Data Parallelism



- The performance and usability issues with distributed TensorFlow can be addressed by adopting an MPI communication model
- Resources dedicated to gradient calculation
- An MPI collective based approach would eliminate the need for PS processes and likely be optimized without intervention from the user
- TensorFlow does have an MPI option, but it only replaces gRPC point to point operations with MPI
  - Collective algorithm optimization in MPI not used

Cray Programming Environment Machine Learning Plugin (CPE ML Plugin)

- **DL communication plugin with Python and C APIs**
- **Optimized for TensorFlow but also portable to other frameworks (testing with PyTorch now)**
  - Callable from C/C++ source
  - Called from Python if data stored in NumPy arrays or Tensors
- **Does not require modification to TensorFlow source**
  - User modifies training script
- **Uses custom allreduce specifically optimized for DL workloads**
  - Optimized for Cray Aries interconnect and IB for Cray clusters
- **Tunable through API and environment variables**
- **Supports multiple gradient aggregations at once with thread teams**
  - Useful for Generative Adversarial Networks (GAN), for example
- **Another alternative is Horovod from Uber**



Alchemist

An Apache Spark ⇔ MPI Interface

A collaboration of Cray (Kristyn Maschhoff and Michael Ringenburg)
and the UC Berkeley RiseLab (Alex Gittens, Kai Rothauge, Michael W. Mahoney, Shusen Wang, and Jey Kottalam)

Slides courtesy Kai Rothauge

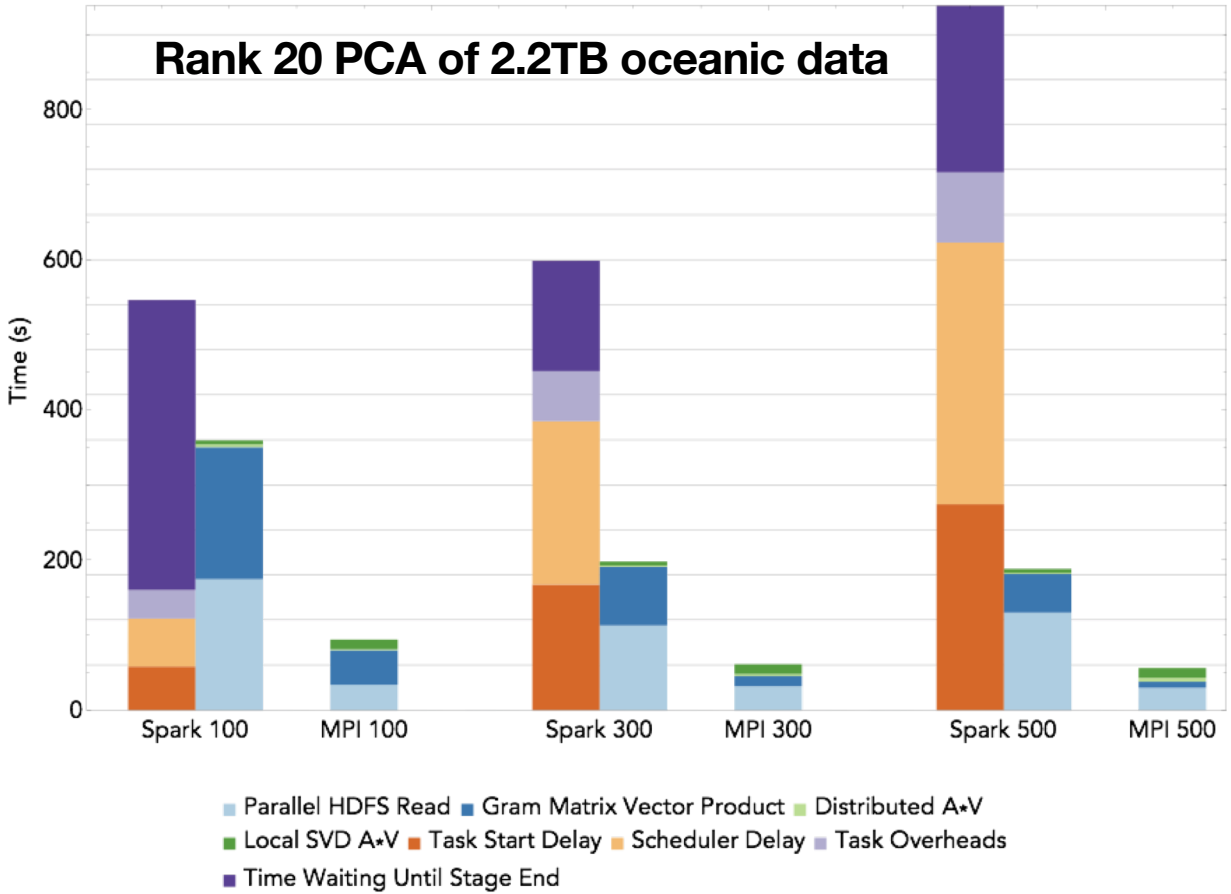
MPI vs Spark

- Cray, NERSC, and AMPLab performed case study for numerical linear algebra on Spark vs. MPI
- Why do linear algebra in Spark?
  - **Pros**:
    - Faster development, easier reuse
    - One abstract uniform interface (RDD)
    - An entire ecosystem that can be used before and after the NLA computations
    - Spark can take advantage of available local linear algebra codes
    - Automatic fault-tolerance, out-of-core support
  - **Con**:
    - Classical MPI-based linear algebra implementations will be faster and more efficient

Slides courtesy Kai Rothauge, UC Berkeley

MPI vs Spark

- Performed a case study for numerical linear algebra on Spark vs. MPI:
  - Matrix factorizations considered include Principal Component Analysis (PCA)
  - Data sets include
    - Oceanic data: 2.2 TB
    - Atmospheric data: 16 TB



A. Gittens et al. "Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+MPI using three case studies", 2016 IEEE International Conference on Big Data (Big Data), pages 204–213, Dec 2016.

Slides courtesy Kai Rothauge, UC Berkeley

MPI vs Spark: Lessons learned

- With favorable data (tall and skinny) and well-adapted algorithms, linear algebra in Spark is 2x-26x slower than MPI when I/O is included
- Spark's overheads are orders of magnitude higher than the actual computations
  - Overheads include time until stage end, scheduler delay, task start delay, executor deserialize time, inefficiencies related to running code via JVM
- **The gaps in performance suggest it may be better to interface with MPI-based codes from Spark**

Slides courtesy Kai Rothauge, UC Berkeley

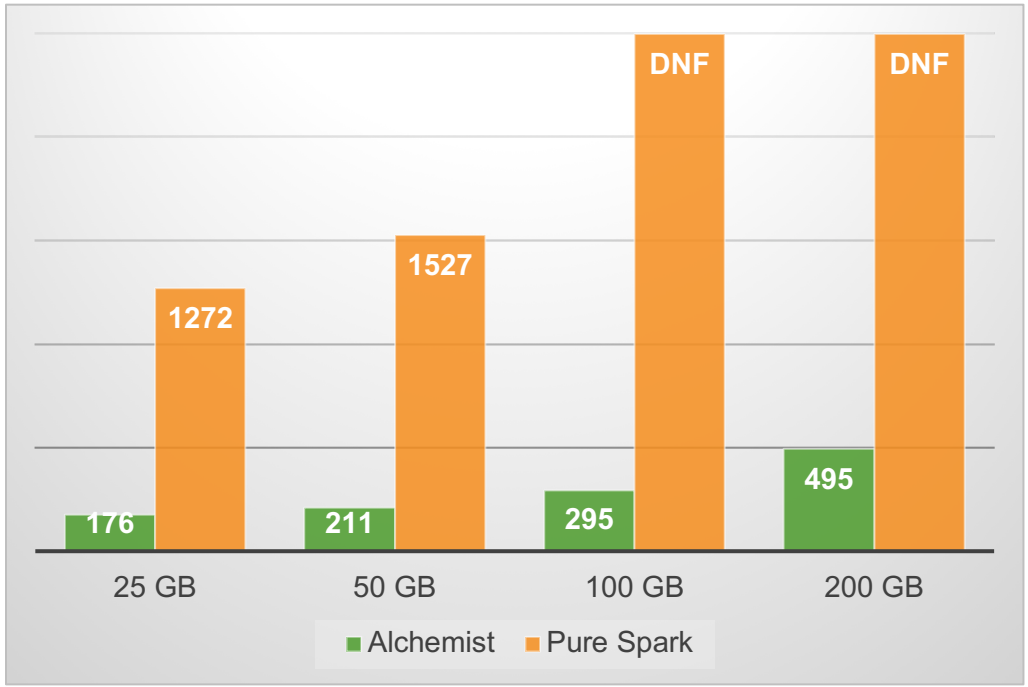
Alchemist

- Interface between Apache Spark and ***existing*** MPI-based libraries for NLA, ML, etc.
- Design goals include making the system *easy to use*, *efficient*, and *scalable*
- Two main tasks:
  - Send distributed **input** matrices from Spark to MPI-based libraries (**Spark => MPI**)
  - Send distributed **output** matrices back to Spark (**Spark <= MPI**)
- Minimize overhead when transferring data between Spark and library

Slides courtesy Kai Rothauge, UC Berkeley

Truncated SVD Alchemist vs Pure Spark

- Use Alchemist and MLlib to get rank 20 truncated SVD
- Setup:
  - 30 KNL nodes, 96GB DDR4, 16GB MCDRAM
  - Spark: 22 nodes; Alchemist: 8 nodes
  - A: m-by-10K, where m = 5M, 2.5M, 1.25M, 625K, 312.5K
  - Ran jobs for at most 60 minutes (3600 s)
- Alchemist times include data transfer



Slides courtesy Kai Rothauge, UC Berkeley

Future Work

- Support for additional MPI-based libraries
  - And make it easier to add new ones
- Enable running on AWS EC2
- Ray + Alchemist?
- More info:
  - CUG 2018: "**Alchemist: An Apache Spark <=> MPI Interface**", Alex Gittens, Kai Rothauge, Shusen Wang, Michael W. Mahoney, Jey Kottalam, Lisa Gerhardt, Prabhat, Michael Ringenburg, Kristyn Maschhoff, https://arxiv.org/abs/1806.01270
  - KDD 2018 (upcoming): "**Accelerating Large-Scale Data Analysis by Offloading to High-Performance Computing Libraries using Alchemist**", Alex Gittens, Kai Rothauge, Shusen Wang, Michael W. Mahoney, Lisa Gerhardt, Prabhat, Jey Kottalam, Michael Ringenburg, Kristyn Maschhoff, https://arxiv.org/abs/1805.11800

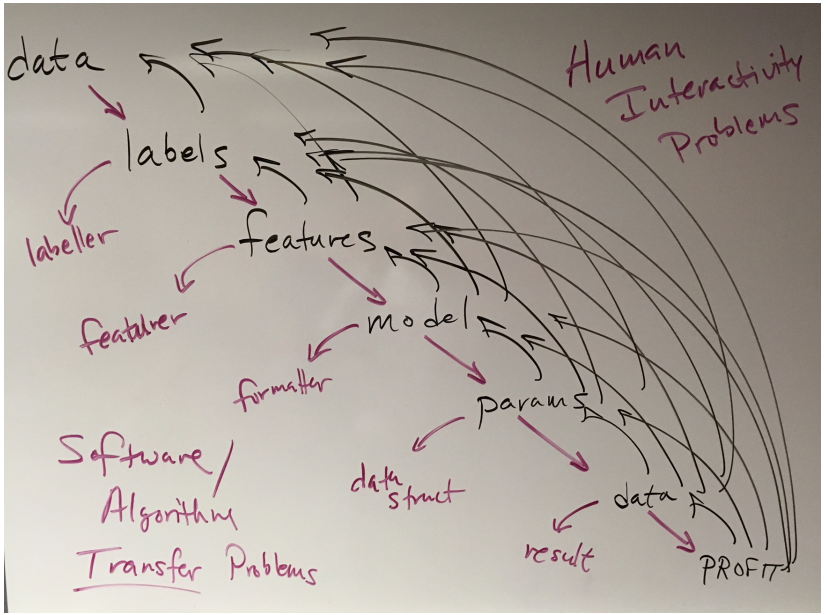
Try it out at github.com/alexgittens/alchemist

Slides courtesy Kai Rothauge, UC Berkeley

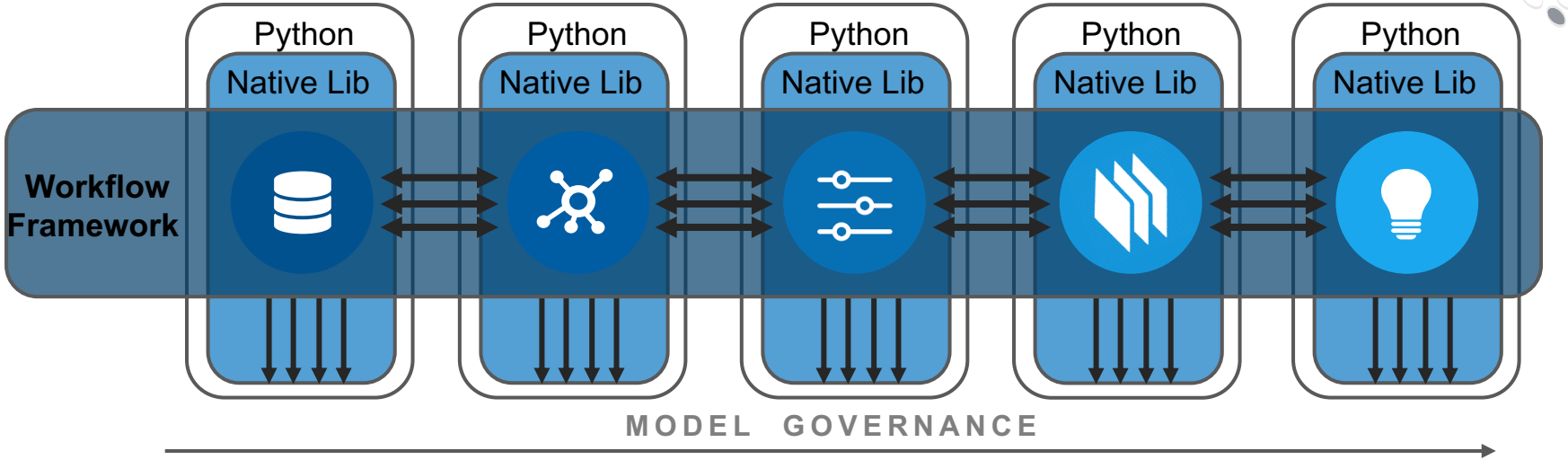
The Future?

End-to-End Data Science Workflows

- **Data science workflows are far more complex than just training and queries**
  - Data preparation
  - Feature selection
  - (Hyper-) parameter tuning
  - Model ensembles
  - Model rationale/interpretability and unlearning
- **Discovering the right workflow more costly than even the largest ML training**
  - ML engineer's 80/20 rule
  - Often iterative/exploratory – requires interactivity/near-real time exploration



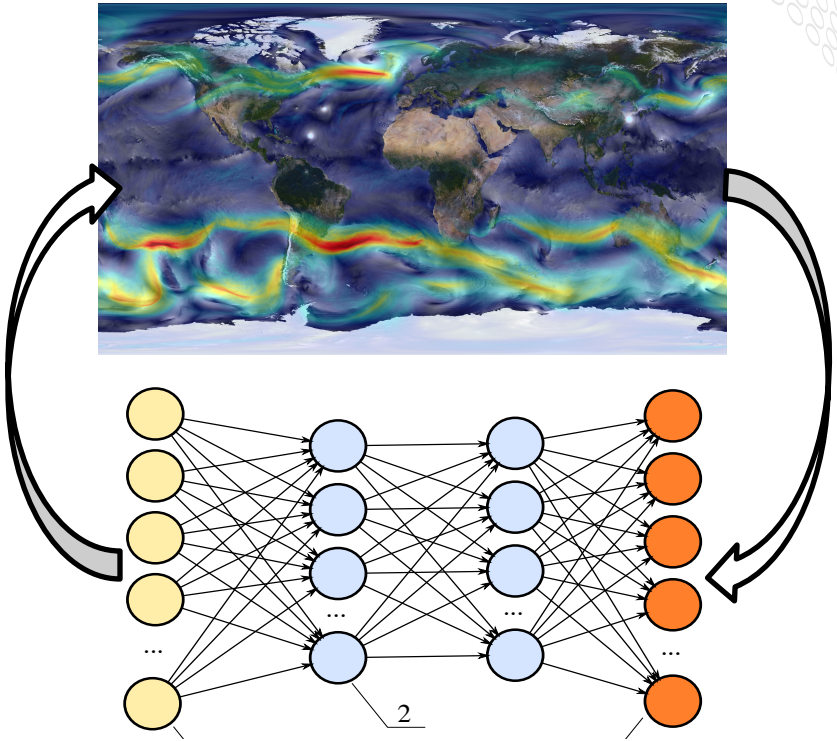
A Vision for Data Science Workflows



- **Complete, integrated, intelligent framework for data science workflows**
  - Modularity and extensibility: Defined, open APIs
  - Ease-of-use: Framework handles back end job and data management
  - Intelligence: Intelligent workflow search strategies
  - Interactive speed: Leverage parallel I/O, interconnect, and compute to allow interactive speed exploration
  - Portability: Run on laptop or supercomputer

Convergence of AI, Analytics, and Simulation

- **How can AI help simulation, and how can simulation help AI?**
  - Trained models to replace expensive computations with “good enough” approximations
  - Training models on simulated results
  - Machine learning to choose optimal simulation parameters (“tuning knobs”)
- **Leverage full capabilities of hardware**
  - Increase utilization
  - Reduce data movement
  - Simplify workflows



Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Questions?